VSFlex 1.0

VideoSoft Custom Control Library

To learn how to use help, press F1

Introduction

Find out about:
Installation, Product support, Licensing, Registration, and Other VideoSoft products.

vsFlexArray

Display and operate on tabular data. FlexArray gives you total flexibility to display, sort, merge, and format tables containing strings and pictures.

Introduction Reference QuickStart

vsFlexString

A powerful regular expression engine. FlexString lets provide regular expression search-and-replace capabilities, parse input strings, and much more.

Introduction Reference QuickStart

Introduction

Welcome to VSFlex 1.0, a VideoSoft Custom Control Library.

VSFlex contains two custom controls designed to save you from writing tedious, repetitive, error-prone code. The controls are innovative and efficient. They are distributed as a single VBX to make installation easier.

Our distribution policy is almost as innovative as the controls. We want every Visual Basic programmer to get a copy of VSFlex and try it for as long as they want. Those who like the product and find it useful (almost everybody, we hope) can buy a license for a reasonable price. The only restriction is that unlicensed copies of VSFlex display a VideoSoft banner whenever they are loaded, to remind developers to license the product.

We hope you'll like VSFlex. If you have suggestions and ideas for new features or new controls, call us or write.

VideoSoft
2625 Alcatraz Avenue, Suite 271
Berkeley, CA 94705
(510) 704-8200 (phone)
(510) 843-0174 (fax)

Control Summary

Icon	Object	Description
	**vsFlexArray**	Display and operate on tabular data. FlexArray gives you total flexibility to display, sort, merge, and format tables containing strings and pictures.
	**vsFlexString**	A powerful regular expression engine. FlexString lets provide regular expression search-and-replace capabilities, parse input strings, and much more.

Installation

To install VSFlex, just copy the following files to your WINDOWS\SYSTEM directory:

VSFLEX.VBX This file contains the controls. To use VSFlex from Visual Basic, you must include this file in your project.

VSFLEX.LIC This is the VSFlex license file. If VSFlex cannot find this file when it starts running, it displays a VideoSoft banner and waits for the user to click Ok, unless the project was compiled on a machine that had VSFLEX.LIC installed.

VSFLEX.HLP This file contains the VSFlex on-line help.

If you would rather install VSFlex in a different directory, thats fine. As long as the help and license files are either in the same directory as the VBX, in the WINDOWS\SYSTEM directory, or in the WINDOWS directory, VSFlex will find them for you.

Distribution

VSFlex is royalty-free. You may include copies of the VBX and HLP files with as many copies of as many applications you ship.

You cannot distribute the license file VSFLEX.LIC. And you dont have to: as long as you have the license file installed on your machine, VSFlex will stamp every application you compile so the banner will not appear when your users run the applications.

If you work with other developers, you may be interested in VideoSofts site licenses. Call us for details.

If you havent yet registered your copy of VSFlex and would like to do it now, click **HERE** to get an Order Form.

Product Support

Product support for VSFlex is available to licensed users through the following channels:

CompuServe	CIS 74774,420 or join our forum by typing **GO VIDEOSOFT**
Mail	VideoSoft 2625 Alcatraz Avenue, Suite 271 Berkeley, California 94705
Phone	(510) 704-8200
Fax	(510) 843-0174

Before calling for technical support, please make sure you know what version of VSFlex you are using. The version number appears in the About box that pops up when you double-click the About property in any of the VSFlex controls.

Also, please make sure you check the last section of the manual, Hints and Troubleshooting. It contains answers to the most common questions people ask our technical support staff. Maybe you can find your answer there.

Using the FlexArray Control

FlexArray is a control that allows you to display and operate on tabular data in a totally new way. On the surface, FlexArray is similar to a spreadsheet, but it allows total flexibility to display, sort, merge, and format tables containing strings and pictures.

The FlexArray control was designed to be compatible with Microsofts Grid control (GRID.VBX). FlexArray implements most of the Grid controls properties, so it is easy to modify older projects to take advantage of FlexArrays extra functionality:

Individual cell formatting: use FlexArrays Cell* properties to control individual cell colors and fonts.

Sorting: use FlexArrays Sort property to sort information with speed and complete flexibility.

Row and Column moving: use FlexArrays RowPosition() and ColPosition() properties to rearrange information at run time.

Cell Merging: use FlexArrays exclusive MergeCells property to create clear, concise, and attractive data summaries without programming.

Design-time layout development: use FlexArrays FormatString property to define column and row headers, widths, and alignment at design-time.

Cell Editing: use FlexArrays CellTop, CellLeft, CellWidth, and CellHeight properties to place controls directly over the current cell, simulating in-cell editing with total program control.

Other FlexArray advantages are: long strings in cells (up to 32k), more than 2000 rows, invisible columns and rows, more control over fonts (see the FontWidth property), more options for customizing colors and grid styles, for aligning text and pictures, less flicker, more control over cursor and selection appearance, and more.

Best of all, the FlexArray is small and does not require separate DLLs, so installation is quick and easy.

FlexArray QuickStart

This section of the manual takes you step-by-step through the creation of three Visual Basic projects using the FlexArray control:

Merge
This sample project illustrates some of FlexArrays sorting and merging capabilities. It shows how FlexArray can be used to implement a display that groups information by category, allowing the user to modify the order in which information is presented.

Edit
This sample project illustrates some of FlexArrays events and container capabilities. It shows how FlexArray can be used to implement a spreadsheet with in-cell editing using standard Visual Basic controls.

Outline
This sample project illustrates some of the FlexArrays cell formatting capabilities. It shows how FlexArray can be used to implement an outline-style display, with heading items that can be collapsed or expanded with the mouse.

Since the sample projects are a little long, you may want to print them (using the File|Print menu option) and work along as you read the code descriptions.

MERGE: Grouping and sorting information

This sample project illustrates some of FlexArrays sorting and merging capabilities. It shows how FlexArray can be used to implement a display that groups information by category, allowing the user to modify the order in which information is presented.

Create Controls

Start a new Visual Basic project including the VSFLEX.VBX file (if you dont know how to add VBX files to a project, consult the Visual Basic manual). The VSFLEX control icons will be added to the Visual Basic toolbox.

Create a FlexArray object on the form by clicking the FlexArray icon on the toolbox, then clicking and dragging on the form.

Set Properties

Click on the FlexArray object you just created by clicking on it, then press F4 to bring up the Properties window. Set the following properties (dont worry about the number of spaces in the **FormatString**):

Name = FA **to save some typing**
Cols = 4
FontName = Arial
FontBold = False
MergeCells = 2 **restrict rows**
FormatString = <Region |<Product |<Employee |>Sales

Make the form wider, if you have to, so the entire control is visible. Your form should look like this:

Notice how the FormatString property affected column headings, alignment, and widths. The pipe character delimits the fields, and the arrows indicate alignment.

Create Data In a real application, your data would probably come from a database. In this sample, we will create some dummy data to work with.

Double-click the form (not the FlexArray) to add code to the Form_Load event. Type in the following:

```
Sub Form_Load ()
    Dim i%

    ' create dummy data
    For i = fa.FixedRows To fa.Rows - 1
      fa.TextArray(fai(i, 0)) = RandString(0)   region
      fa.TextArray(fai(i, 1)) = RandString(1)   product
      fa.TextArray(fai(i, 2)) = RandString(2)   employee
      fa.TextArray(fai(i, 3)) = Format(Rnd * 10000, "#.00")
    Next

    ' set up merging
    fa.MergeCol(0) = True
    fa.MergeCol(1) = True
    fa.MergeCol(2) = True

    ' sort to see the effects
    DoSort
End Sub
```

You cant run this project yet, because we have to define a couple of general routines. The first one calculates an index to be used with the TextArray property. Here it is:

```
Function fai (r%, c%) As Integer
    fai = c + fa.Cols * r
End Function
```

For details on how this works, see the TextArray property in the reference section of the manual.

The second routine sorts all the random data:

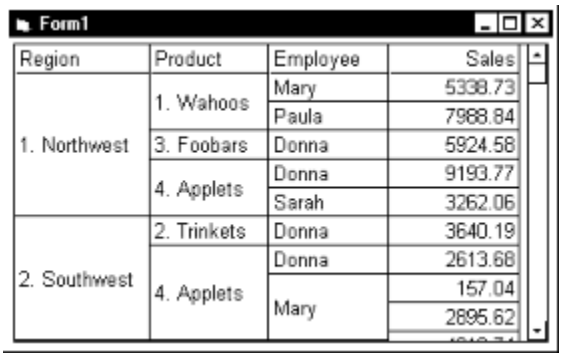
```
Sub DoSort ()
    fa.Col = 0
    fa.ColSel = fa.Cols - 1
    fa.Sort = 1  generic ascending
End Sub
```

The routine selects an entire row, then uses the Sort property to sort the entire grid in ascending order based on columns 0, 1, 2, and 3. Notice that the generic sort option works for strings and numbers.

Finally, heres the function that returns random strings to fill up the FlexArray with dummy data:

```
Sub RandomString (kind%)
    Select Case kind

      Case 0   region
        Select Case (Rnd * 1000) Mod 5
          Case 0: Region = "1. Northwest"
          Case 1: Region = "2. Southwest"
          Case 2: Region = "3. Midwest"
          Case 3: Region = "4. East"
          Case Else: Region = "5. Overseas"
        End Select

      Case 1   product
       Select Case (Rnd * 1000) Mod 5
            Case 0: Product = "1. Wahoos"
            Case 1: Product = "2. Trinkets"
            Case 2: Product = "3. Foobars"
            Case Else: Product = "4. Applets"
        End Select

      Case 2   employee
        Select Case (Rnd * 1000) Mod 4
          Case 0: Employee = "Mary"
          Case 1: Employee = "Sarah"
          Case 2: Employee = "Donna"
          Case Else: Employee = "Paula"
      End Select
    End Select
End Sub
```

Were almost done. If you run the project, it should look like this:



Pretty neat, huh? You can see at a glance which products sold in each region, and who sold them.

Dynamic Layout

Were not done yet. Why not let the user switch views, to see for example which products are being sold by which employees? We only need two very short routines to do this:

```
Sub fa_MouseDown (Button%, Shift%, X!, Y!)
    fa.Tag = ""
    If fa.MouseRow <> 0 Then Exit Sub
    fa.Tag = Str(fa.MouseCol)
    MousePointer = 9
End Sub
```

This routine checks when the user presses the mouse button over the heading row, and uses the Tag property to save the column number. (This way, we dont have to declare any global variables.) We also change the cursor to give the user some feedback.

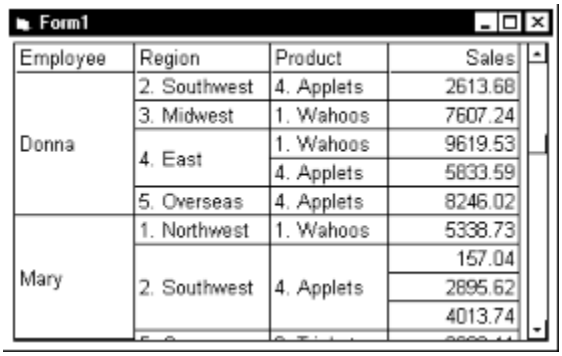
And finally, we need to trap the MouseUp event as well:

```
Sub fa_MouseUp (Button%, Shift%, X!, Y!)
    MousePointer = 0
    If fa.Tag = "" Then Exit Sub
    fa.Redraw = False
    fa.ColPosition(Val(fa.Tag)) = fa.MouseCol
    DoSort
    fa.Redraw = True
End Sub
```

This routine first restores the cursor, then checks to see if the Tag property currently holds a valid column number. If it does, then we set Redraw to False to reduce flicker while we work.

Next, we use the ColPosition property to move the selected column to the desired position, and call DoSort to reorganize the data. Finally, we set the Redraw property back to True so the FlexArray shows the data in its new state.

Try running the project now. Drag the Employee column over to the left, and the Product column next to it. The FlexArray will look like this:



Its not just data anymore. Its information.

EDIT In-cell editing

This sample project illustrates some of FlexArrays events and container capabilities. It shows how FlexArray can be used to implement a spreadsheet with in-cell editing using standard Visual Basic controls.

Create Controls

Start a new Visual Basic project including the VSFLEX.VBX file (if you dont know how to add VBX files to a project, consult the Visual Basic manual). The VSFLEX control icons will be added to the Visual Basic toolbox.

Create a FlexArray object on the form by clicking the FlexArray icon on the toolbox, then clicking and dragging on the form.

Now create a Text Box, but **dont place it on the form**. Place it on the FlexArray instead. To make sure you got it right, try to drag the text box off the FlexArray. If you cant, that means its really on the FlexArray, and youre Ok. If you can, delete it and try again.

Set Properties

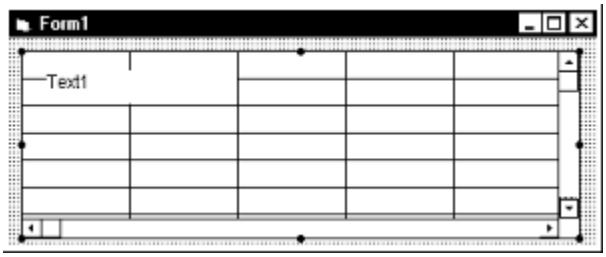
Click on the FlexArray object you just created by clicking on it, then press F4 to bring up the Properties window. Set the following properties (dont worry about the number of spaces in the FormatString):

Name = FA to save some typing
FontName = Arial
FontSize = 9
FontBold = False
FillStyle = 1 repeat
FocusRect = 2 heavy

Now click on the Text Box and set the following properties:

Name = Text
FontName = Arial
FontSize = 9
FontBold = False
BorderStyle = 0 none
Visible = False

Your form should look like this:



Add Row and Column Headings

To make the FlexArray look like a spreadsheet, double-click on the form and attach the following code to the Form_Load event:

```
Sub Form_Load ()
    Dim i%

    ' make first column narrow
    fa.ColWidth(0) = fa.ColWidth(0) / 2
    fa.ColAlignment(0) = 1    center center

    ' label rows and columns
    For i = fa.FixedRows To fa.Rows - 1
        fa.TextArray(fai(i, 0)) = i
    Next
```

```
    For i = fa.FixedCols To fa.Cols - 1
        fa.TextArray(fai(0, i)) = i
    Next

    ' initialize edit box (so it loads now)
    text = ""
End Sub
```

When the form loads, we set the width of the first column to half of its default width, and make center-align its contents. Next, we fill the header rows and columns with labels to identify the cells. To do this, we use the TextArray property and the same helper function we had used earlier to calculate an index for the TextArray property. Here it is again:

```
Function fai (r%, c%) As Integer
    fai = c + fa.Cols * r
End Function
```

If you run the project now, you will have a spreadsheet with no editing capability. You can navigate around, though, and select ranges using the keyboard or the mouse.

Add In-Cell Editing

We are ready to add editing capabilities to the project. Lets say we want to bring up an edit box whenever the user starts typing or double-clicks on a cell.

We start by adding the following code:

```
Sub fa_KeyPress (KeyAscii%)
    FlexArrayEdit fa, Text, KeyAscii
End Sub

Sub fa_DblClick ()
    FlexArrayEdit fa, Text, 32    simulate a space
End Sub
```

The FlexArrayEdit is a really useful routine, so we should place it in a separate module so we can reuse it in other projects. For now, well define it like this:

```
Sub FlexArrayEdit (FlexArray As Control, Edt As Control,
                   KeyAscii%)

  ' use the character that was typed
  Select Case keyascii

    ' a space means edit the current text
    Case 0 To 32
      Edt = FlexArray
      Edt.SelStart = 1000

    ' anything else means replace the current text
    Case Else
      Edt = Chr(keyascii)
      Edt.SelStart = 1
  End Select

  ' show Edt at the right place
  Edt.Move FlexArray.CellLeft, FlexArray.CellTop
  Edt.Width = FlexArray.CellWidth
  Edt.Height = FlexArray.CellHeight
  Edt.Visible = True

  ' and let it work
  Edt.SetFocus
End Sub
```

This routine takes care of initializing the Text box at the right spot and passing it the focus, so the user can type into it. It would be easy to extend or create variations using Combo boxes or other controls.

Now we need to add some code to the Edit box so it knows how to update the data and when to return control to the FlexArray. Again, we call a generic function that can be reused. Here it is:

```
Sub Text_KeyPress (KeyAscii%)
      eat returns to get rid of beep
    If KeyAscii = 13 Then KeyAscii = 0
End Sub

Sub Text_KeyDown (KeyCode%, Shift%)
    EditKeyCode fa, text, KeyCode, Shift
End Sub

Sub EditKeyCode (FlexArray As Control, Edt As Control,
                 KeyCode%, Shift%)

  ' standard edit control processing
  Select Case KeyCode

    Case 27 ' ESC: hide, return focus to FlexArray
      Edt.Visible = False
      FlexArray.SetFocus

    Case 13 ' ENTER return focus to FlexArray
      FlexArray.SetFocus

    Case 38 ' up
      FlexArray.SetFocus
      DoEvents
      If FlexArray.Row > FlexArray.FixedRows Then
        FlexArray.Row = FlexArray.Row - 1
      End If

    Case 40 ' down
      FlexArray.SetFocus
      DoEvents
      If FlexArray.Row < FlexArray.Rows - 1 Then
        FlexArray.Row = FlexArray.Row + 1
      End If
  End Select
End Sub
```

Ok, now the editor knows how to handle the keyboard and return the focus to the FlexArray. There is only one last thing we have to do: tell the FlexArray what to do with the information when it becomes available.

We need to handle only two situations: when the Edit control voluntarily returns the focus to the FlexArray, and when the user clicks on a different cell thereby activating the FlexArray.

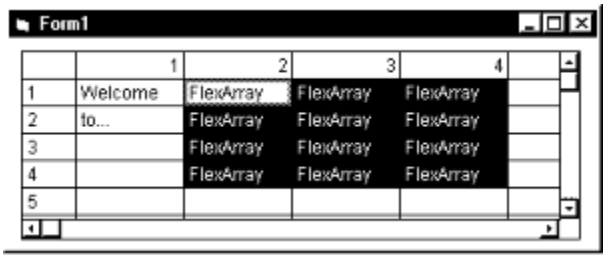
In both situations, what we have to do is check whether the text box is up. If it is, copy the information from it and hide it. Here is the code:

```
Sub fa_GotFocus ()
    If Text.Visible = False Then Exit Sub
    fa = Text
    Text.Visible = False
End Sub

Sub fa_LeaveCell ()
    If Text.Visible = False Then Exit Sub
    fa = Text
    Text.Visible = False
End Sub
```

Were done. In a real application, the above routines could include all kinds of data validation, and refuse to copy the contents of the Edit box into the FlexArray in case errors were detected.

Run the project, and type something into the cells. Notice that you can select a range and fill it

easily, or use the ESC key to cancel changes while you are editing a cell. Heres what the program looks like:



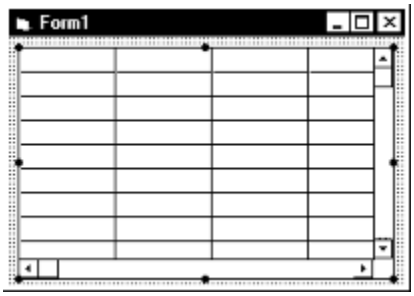
OUTLINE: An outline-style expense report

This sample project illustrates some of the FlexArrays cell formatting capabilities. It shows how FlexArray can be used to implement an outline-style display, with heading items that can be collapsed or expanded with the mouse.

Create Controls

Start a new Visual Basic project including the VSFLEX.VBX file (if you dont know how to add VBX files to a project, consult the Visual Basic manual). The VSFLEX control icons will be added to the Visual Basic toolbox.

Create a FlexArray object on the form by clicking the FlexArray icon on the toolbox, then clicking and dragging on the form. Your form should look like this:

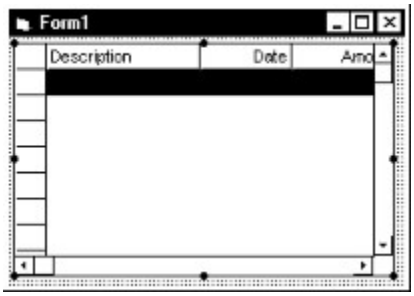


Set Properties

Click on the FlexArray object you just created by clicking on it, then press F4 to bring up the Properties window. Set the following properties (dont worry about the number of spaces in the FormatString):

```
Name = FA            to save some typing
Cols = 4
FontName = Arial
FontBold = False
GridLines = 0        none
SelectionMode = 1    by row
FocusRect = 0        none
FillStyle = 1        repeat
FormatString = ^      |Description                |
                  >Date                  |>Amount
```

Make the form wider, if you have to, so the entire control is visible. Your form should look like this:



Notice how the FormatString property affected column headings, alignment, and widths. The pipe character delimits the fields, and the arrows indicate alignment.

Create Data

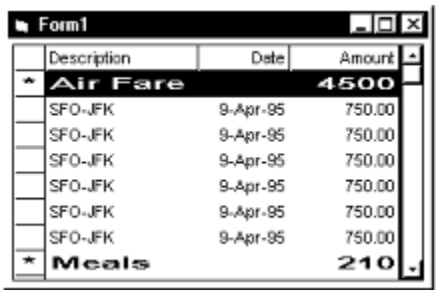
In a real application, your data would come from a database or from the user. In this sample, we will create some dummy data to work with.

Double-click the form (not the FlexArray) to add code to the Form_Load event. Type in the following:

```
Sub Form_Load ()
   Dim i%, tot%
   Dim t$, s$
```

```
  create some dummy data
t = Chr(9)
fa.Rows = 1

fa.AddItem "*" + t + "Air Fare"
s = "" +t+ "SFO-JFK" +t+ "9-Apr-95" +t+ "750.00"
For i = 0 to 5
   fa.AddItem s
Next

fa.AddItem "*" + t + "Meals"
s = "" +t+ "Flint's BBQ" +t+ "25-Apr-95" +t+ "35.00"
For i = 0 to 5
   fa.AddItem s
Next

fa.AddItem "*" +t+ "Hotel"
s = "" +t+ "Center Plaza" +t+ "25-Apr-95" +t+ "817.00"
For i = 0 to 5
   fa.AddItem s
Next

  add up totals and format heading entries
For i = fa.Rows - 1 To 0 Step -1
   If fa.TextArray(i * fa.Cols) = "" Then
      tot = tot + Val(fa.TextArray(i * fa.Cols + 3))
   Else
      fa.Row = i
      fa.Col = 0
      fa.ColSel = fa.Cols - 1
      fa.CellBackColor = &HC0C0C0
      fa.CellFontBold = True
      fa.CellFontWidth = 8
      fa.TextArray(i * fa.Cols + 3) = Format(tot, "0")
      tot = 0
   End If
Next

  select the first row
fa.Col = 1
fa.Row = 1
fa.ColSel = fa.Cols - 1
End Sub
```

Sorry for all the typing, but now were almost done. If you run the project, it should look like this:



Notice how the special formatting sets off the headings.

Collapse/Expand

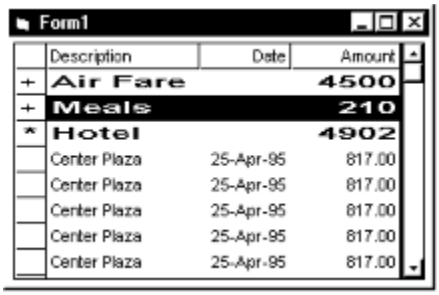
Now we are ready to add the code that collapses and expands headings. Stop the program, double-click the FlexArray, and add the following code to the FlexArrays DblClick event:

```
Sub fa_DblClick ()
```

```
    Dim i%, r%

     ignore top row
    r = fa.MouseRow
    If r < 1 Then Exit Sub

    ' find field to collapse or expand
    While r > 0 And fa.TextArray(r * fa.Cols) = ""
        r = r - 1
    Wend

    ' show collapsed/expanded symbol on first column
    If fa.TextArray(r * fa.Cols) = "*" Then
        fa.TextArray(r * fa.Cols) = "+"
    Else
        fa.TextArray(r * fa.Cols) = "*"
    End If

    ' expand items under current heading
    r = r + 1
    If fa.RowHeight(r) = 0 Then
        Do While fa.TextArray(r * fa.Cols) = ""
            fa.RowHeight(r) = -1   default row height
            r = r + 1
            If r >= fa.Rows Then Exit Do
        Loop

    ' collapse items under current heading
    Else
        Do While fa.TextArray(r * fa.Cols) = ""
            fa.RowHeight(r) = 0   hide row
            r = r + 1
            If r >= fa.Rows Then Exit Do
        Loop
    End If
End Sub
```

Thats it. Notice how setting the RowHeight() property to -1 resets it to the default height, determined by the font size, and setting it to 0 hides the entire row.

Run the project and double-click on the rows to expand and collapse headings. Your form should look like this:



You could easily modify this project to show pictures instead of + and * characters, or add additional levels to the outline.

FlexArray Control Reference

Description A FlexArray control displays a series of rows and columns. The intersection of a row and column is a cell. You can read and set the contents of each cell programmatically.

Remarks You can put text, a picture, or both in any cell of a FlexArray. The Row and Col properties specify the current cell in a grid. You can specify the current cell in code, or the user can change it at run time using the mouse or the arrow keys. The Text property references the contents of the current cell.

If a cell's text is too long to be displayed in the cell, and the WordWrap property is set to True, the text wraps to the next line within the same cell. To display the wrapped text, you may need to increase the cell's column width (ColWidth property) or row height (RowHeight property).

Use the Cols and Rows properties to determine the number of columns and rows in a FlexArray control.

When a new element of a control array is loaded at run time, the new element does not inherit the original control's run-time properties.

File Name VSFLEX.VBX

Object Type FlexArray

Note Before you can use a FlexArray control in your application, you must add VSFLEX.VBX to your project (see the Visual Basic manual for details). To automatically include VSFLEX.VBX in new projects, put it in an AUTOLOAD.MAK file. When distributing your application, you should install the VSFLEX.VBX file in the user's Microsoft Windows SYSTEM subdirectory.

FlexArray Summary

Properties (default: Text)

* (About)	Align	BackColor
* BackColorBkg	* BackColorFixed	* BackColorSel
BorderStyle	* CellAlignment	* CellBackColor
* CellFontBold	* CellFontItalic	* CellFontName
* CellFontSize	* CellFontWidth	* CellForeColor
* CellHeight	* CellLeft	* CellPicture
* CellPictureAlignment	* CellTextStyle	* CellTop
* CellWidth	* Clip	* Col
* ColAlignment()	* ColData()	* ColPosition()
* Cols	* ColSel	* ColWidth()
DragIcon	DragMode	Enabled
* FillStyle	* FixedCols	* FixedRows
* FocusRect	FontBold	FontItalic
FontName	FontSize	FontStrike
FontUnder	* FontWidth	ForeColor
* ForeColorFixed	* ForeColorSel	* FormatString
* GridColor	* GridColorFixed	* GridLines
* GridLinesFixed	Height	HelpContextID
* HighLight	hWnd	Index
Left	* LeftCol	* MergeCells
* MergeCol()	* MergeRow()	* MouseCol
MousePointer	* MouseRow	Name
Parent	* Redraw	* Row
* RowData()	* RowHeight()	* RowHeightMin
* RowPosition()	* Rows	* RowSel
* SelectionMode	* ScrollBars	* ScrollTrack
* Sort	TabIndex	TabStop
Tag	* Text	* TextArray()
* TextStyle	* TextStyleFixed	Top
* TopRow	* Version	Visible
Width	* WordWrap	

Events

Click	DblClick	DragDrop
DragOver	* EnterCell	GotFocus
KeyDown	KeyPress	KeyUp
* LeaveCell	LostFocus	MouseDown
MouseMove	MouseUp	* RowColChange
* Scroll	* SelChange	

Methods

* AddItem	* Clear	Drag
LinkSend	Move	Refresh
* RemoveItem	ZOrder	

AddItem Method

Description Adds a new row to a grid control at run time.

Syntax [form!]FlexArray.**AddItem** *item* [, *index*]

Remarks The AddItem method has these parts:

item
String expression to add to the control. Use the tab character (character code 9) to separate multiple strings you want inserted into each column of a newly added row.

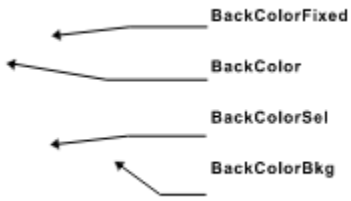
index
Integer representing the position within the control where the new row is placed. For the first row, index = 0. If *index* is omitted, the new row becomes the last.

BackColorBkg, BackColorFixed, BackColorSel Properties

Description Determine the background color of various elements of the FlexArray.

Usage [form!]FlexArray.**BackColorBkg**[= *colorexpression*]

Remarks The picture below shows what part of the FlexArray each property refers to:



To set the background color of individual cells, use the CellBackColor property.

Data Type Long (Color)

CellAlignment Property

Description Determines the alignment of data in a cell or range of selected cells. Not available at design time.

Usage [form!]FlexArray.**CellAlignment**[= *numericexpression*]

Setting The CellAlignment property settings are:

0 - Left Top	3 - Center Top	6 - Right Top
1 - Left Center	4 - Center Center	7 - Right Center
2 - Left Bottom	5 - Center Bottom	8 - Right Bottom

9 - General (default: Left Center for strings and Right Center for numbers)

Remarks Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

To set the alignment of entire columns, use the ColAlignment property.

To set column alignments at design time, use the FormatString property.

Data Type Integer (Enumerated)

CellBackColor, CellForeColor Properties

Description Determines the background and foreground colors of individual cells or ranges of cells.

Usage [form!]FlexArray.**CellBackColor**[= *colorexpression*]

Remarks Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

Setting either of these properties to zero causes the FlexArray to paint the cell using the standard background and foreground colors. If you want to set either of these properties to black, set them to one instead of zero.

To set the colors of various FlexArray elements, use the BackColor* and ForeColor* properties.

Data Type Long (Color)

CellFont* Properties

Description Determine the font to be used for individual cells or ranges of cells.

Usage [form!]FlexArray.**CellFont***[= *fontspec*]

Remarks Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

Data Type String (CellFontName)
Single (CellFontSize, CellFontWidth)
Boolean (CellFontBold, CellFontItalics)

CellForeColor, CellBackColor Properties

Description Determines the background and foreground colors of individual cells or ranges of cells.

Usage [form!]FlexArray.**CellBackColor**[= *colorexpression*]

Remarks Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

Setting either of these properties to zero causes the FlexArray to paint the cell using the standard background and foreground colors. If you want to set either of these properties to black, set them to one instead of zero.

To set the colors of various FlexArray elements, use the BackColor* and ForeColor* properties.

Data Type Long (Color)

CellHeight, CellLeft, CellTop, CellWidth Properties

Description Determine the position of the current cell, in Twips.

Usage *variable* = [form!]FlexArray.**CellTop**

Remarks These properties are useful if you want to emulate in-cell editing. By trapping the FlexArrays KeyPress event, you can place a text box or some other control over the current cell and let the user edit its contents.

See the FlexArray demo on the distribution diskette for an example that uses text boxes and combo boxes to edit cells.

Whenever you read any of these properties, the FlexArray assumes that you want to work on the current cell and it automatically brings it into view, scrolling if necessary.

These properties are read-only.

Data Type Long

CellPicture Property

Description Determines a graphic to be displayed in the current cell or in a range of cells.

Usage [form!].FlexArray.**CellPicture**[= *picture*]

Remarks You can set this property at run time using the LoadPicture function on a bitmap, icon, or metafile, or by assigning to it another controls Picture property.

Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

Each cell may contain text and a picture. The relative position of the text and picture is determined by the CellAlignment and CellPictureAlignment properties.

Data Type Picture

CellPictureAlignment Property

Description Determines the alignment of pictures in a cell or range of selected cells. Not available at design time.

Usage [form!]FlexArray.CellPictureAlignment[= *numericexpression*]

Setting The CellPictureAlignment property settings are:

0 - Left Top	3 - Center Top	6 - Right Top
1 - Left Center	4 - Center Center	7 - Right Center
2 - Left Bottom	5 - Center Bottom	8 - Right Bottom
9 - Stretch	10 - Tile	

Remarks Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

See also the CellPicture property.

Data Type Integer (Enumerated)

CellTextStyle Property

Description Determines 3D effects for text on a specific cell or range of cells.

Usage [form!]FlexArray.**CellTextStyle** [= *setting*]

Remarks The CellTextStyle property settings are:

0 - Flat
1 - Raised
2 - Inset
3 - Raised Light
4 - Inset Light

Settings 1 and 2 work best for large and bold fonts. Settings 3 and 4 work best for small regular fonts.

Changing this property affects the current cell or the current selection, depending on the setting of the FillStyle property.

See also the TextStyle and TextStyleFixed properties.

Data Type Integer

CellTop, CellWidth, CellHeight, CellLeft Properties

Description Determine the position of the current cell, in Twips.

Usage *variable* = [form!]FlexArray.**CellTop**

Remarks These properties are useful if you want to emulate in-cell editing. By trapping the FlexArrays KeyPress event, you can place a text box or some other control over the current cell and let the user edit its contents.

See the FlexArray demo on the distribution diskette for an example that uses text boxes and combo boxes to edit cells.

Whenever you read any of these properties, the FlexArray assumes that you want to work on the current cell and it automatically brings it into view, scrolling if necessary.

These properties are read-only.

Data Type Long

Clear Method

Description Clears the contents of the FlexArray. This includes all text, pictures, and cell formatting.

Syntax [form!]FlexArray.**Clear**

Remarks The Clear method does not affect the number of rows and columns on the FlexArray.

Clip Property

Description Determines the contents of the cells in a FlexArray's selected region. Not available at design time.

Usage [form!]FlexArray.**Clip**[= *stringexpression*]

Remarks The *stringexpression* can contain the contents of multiple rows and columns. In *stringexpression*, a tab character (ANSI character 9) indicates a new cell in a row, and a carriage return (ANSI character 13) indicates the beginning of a new row. Use the Chr function to embed these characters in strings. For example, this code puts text into a selected area two rows high and two columns wide:

```
S$ = "1st" & Chr(9) & "a" & Chr(13) & "2nd" & Chr(9) & "b"
FlexArray.Clip = S$
```

When placing data into a grid, only the selected cells are affected. If there are more cells in the selected region than are described in *stringexpression*, the remaining cells are left alone. If there are more cells described in *stringexpression* than in the selected region, the unused portion of *stringexpression* is ignored.

Data Type String

Col, Row Properties

Description Determine the active cell in a FlexArray. Not available at design time.

Usage [form!]FlexArray.**Col**[= *colnum*]
[form!]FlexArray.**Row**[= *rownum*]

Remarks Use these properties to specify a cell in a FlexArray or to find out which row or column contains the current cell. Columns and rows are numbered from zero, beginning at the top for rows and at the left for columns.

Setting these properties automatically resets RowSel and ColSel, so the selection becomes the current cell. Therefore, to specify a block selection, you must set Row and Col first, then set RowSel and ColSel.

Note The Col, Row properties are not the same as the Cols, Rows properties.

Data Type Integer

ColAlignment() Property

Description Determines the alignment of data in a column. Not available at design time (except indirectly through the FormatString property).

Usage [form!]FlexArray.**ColAlignment**(*colnum*)[= *numericexpression*]

Setting The ColAlignment property settings are:

0 - Left Top	3 - Center Top	6 - Right Top
1 - Left Center	4 - Center Center	7 - Right Center
2 - Left Bottom	5 - Center Bottom	8 - Right Bottom

9 - General (default: Left Center for strings and Right Center for numbers)

Remarks Any column can have an alignment that is different from other columns. This property affects all cells in the specified column, including those in fixed rows.

If *colnum* is -1, then the FlexArray assumes you want to set the alignment of all columns at once.

To set individual cell alignments, use the CellAlignment property.

To set column alignments at design time, use the FormatString property.

Data Type Integer (Enumerated)

ColData(), RowData() Properties

Description These properties are arrays of long integer values with one item for each row (RowData) and for each column (ColData) of the FlexArray. Not available at design time.

Usage [form!]FlexArray.**RowData**(*rownum*)[= *numericexpression*]
[form!]FlexArray.**ColData**(*colnum*)[= *numericexpression*]

Remarks Use the RowData() and ColData() properties to associate a specific number with each row or column on a FlexArray. You can then use these numbers in code to identify the items.

For example, you can add rows containing totals to a FlexArray and identify those rows by setting their RowData() property to a non-zero value. To update the totals later, you can delete the old totals by scanning the RowData() array and removing the appropriate rows.

Another typical use of the RowData property is to keep an index into an array of data structures associated with the items described on each row.

Data Type Long integer

ColPosition(), RowPosition() Properties

Description Allow you to move rows and columns to specific positions on the FlexArray.

Usage [form!]FlexArray.**ColPosition(***colnum***)**[= *newcolnum*]
[form!]FlexArray.**RowPosition(***rownum***)**[= *newrownum*]

Remarks The index and setting must correspond to valid row or column numbers (in the range 0 to Rows - 1 or Cols - 1) or an error will be generated.

For example, the following code moves a column to first position when the user clicks on it:

```
Sub FlexArray_Click ()
   FlexArray.ColPosition(FlexArray.MouseCol) = 0
End Sub
```

When a row or column is moved with these properties, all formatting information moves with it, including width, height, alignment, colors, fonts, etc. To move text only, use the Clip property instead.

Data Type Integer

Cols, Rows Properties

Description Determine the total number of columns or rows in a FlexArray. The minimum is 0. The maximum number of rows and columns is limited by the memory available on your computer.

Usage [form!]FlexArray.**Cols**[= *numericexpression*]
[form!]FlexArray.**Rows**[= *numericexpression*]

Remarks You can use these properties to expand and shrink a FlexArray dynamically at run time.

Note The Cols, Rows properties are not the same as the Col, Row properties.

Data Type Integer

ColSel, RowSel Properties

Description Determine the starting or ending row or column for a range of cells. Not available at design time.

Usage [form!]FlexArray.**RowSel**[= *numericexpression*]
[form!]FlexArray.**ColSel**[= *numericexpression*]

Remarks You can use these properties to select a specific region of the FlexArray from code, or to read into code the dimensions of an area that the user selects.

The FlexArray cursor it the cell at **Row**, **Col**. The FlexArray selection is the region between rows **Row** and **RowSel** and columns **Col** and **ColSel**. Note that **RowSel** may be above or below **Row**, and **ColSel** may be to the left or to the right of **Col**.

Whenever you set the **Row** and **Col** properties, **RowSel** and **ColSel** are automatically reset so the cursor becomes the current selection. If you want to select a block of cells from code, you must set the **Row** and **Col** properties first, then set **RowSel** and **ColSel**.

Data Type Integer

ColWidth() Property

Description Determines the width of the specified column in Twips. Not available at design time.

Usage [form!]FlexArray.**ColWidth**(*colnum*)[= *numericexpression*]

Remarks You can use this property to set the width of any column at run time. For instructions on setting column widths at design-time, see the FormatString property.

You can set ColWidth to zero to create invisible columns, or to -1 to reset the column width to its default value, which depends on the size of the current font.

If *colnum* is -1, then the FlexArray assumes you want to set the width of all columns at once.

Data Type Long

EnterCell Event

Description Occurs when the currently active cell changes to a different cell.

Syntax Sub FlexArray_**EnterCell** ( )

Remarks This event occurs whenever the user clicks a cell other than the selected cell or when you programmatically change the active cell within a selection.

See also the LeaveCell event.

FillStyle Property

Description Determines whether setting the Text property or one of the Cell formatting properties of a FlexArray applies the change to all selected cells.

Usage [form!]FlexArray.**FillStyle**[= *style*]

Setting The FillStyle property settings are:

0 - Single	Changing the Text/Cell* properties only affects the active cell.
1 - Repeat	Changing the Text/Cell* properties affects all selected cells.

Data Type Integer (Enumerated)

FixedCols, FixedRows Properties

Description Determine the total number of fixed columns or fixed rows for a FlexArray. By default, a FlexArray has one fixed column and one fixed row.

Usage [form!]FlexArray.**FixedCols**[= *numericexpression*]
[form!]FlexArray.**FixedRows**[= *numericexpression*]

Remarks A fixed column is a stationary column on the left side of the FlexArray. A fixed row is a stationary row along the top of the FlexArray. You can have zero or more fixed columns and zero or more fixed rows. Fixed columns and rows do not move when the other columns or rows in the grid are scrolled. You can select the colors, font, grid and text style use for the fixed columns and rows.

Fixed columns and rows are typically used in spreadsheet applications to display row numbers and column names or letters.

Data Type Integer

FocusRect Property

Description Determines whether the FlexArray control should draw a focus rectangle around the current cell.

Usage [form!]FlexArray.**FocusRect**[= *setting*]

Setting The FocusRect property settings are:

0 - None
1 - Light (default)
2 - Heavy

Remarks If a focus rectangle is drawn, then the current cell is painted in the background color, as in most spreadsheets and grids. Otherwise, the current cell is painted in the selection color, so you can see which cell is selected even without the focus rectangle.

Data Type Integer (Enumerated)

FontWidth Property

Description Determines the width of the font to be used for text displayed in a FlexArray.

Usage [form!]FlexArray.**FontSize**[= *points*]

Remarks The font width is normally chosen by Windows to match the selected font height and provide a standard aspect ratio. FlexArray allows you to specify fonts that are narrower or wider than the default so you can display more information in a cell or highlight certain cells.

When you specify a font width, Windows will try to select or generate a font to match your request. For best results, use TrueType fonts, which are more flexible. The Courier New font, for instance, looks very good when you make it a little narrower than its default.

To restore the default font width, set this property to zero.

To set the font of individual cells or cell ranges, use the CellFont* properties.

Data Type Single

ForeColorFixed, ForeColorSel Properties

Description Determine the color used to draw text on each part of the FlexArray.

Usage [form!]FlexArray.**ForeColorFixed**[= *colorexpression*]

Remarks The picture below shows what part of the FlexArray each property refers to:

To set the text color of individual cells, use the CellForeColor property.

Data Type Long (Color)

FormatString Property

Description Allows you to set up a FlexArrays column widths, alignments, and fixed row and column text at design time.

Usage [form!]FlexArray.**FormatString**[= *string*]

Remarks FlexArray parses the FormatString at design time and interprets it to get the following information: number of rows and columns, text for row and column headings, column width, and column alignment.

The FormatString is made up of segments separated by pipe characters (|). The text between pipes defines a column, and it may contain the special alignment characters <, ^, or >, to align the entire column to the left, center, or right. The text is assigned to row zero, and its width defines the width of each column.

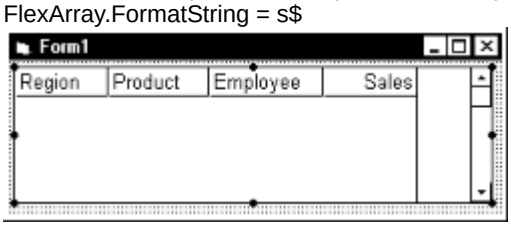
The FormatString may also contain a semi-colon (;), which causes the remaining of the string to be interpreted as row heading and width information. The text is assigned to column zero, and the longest string defines the width of column zero.

FlexArray will create additional rows and columns to accommodate all fields defined by the FormatString, but it will not delete rows or columns if only a few fields are specified. If you want, you can do this by setting the Rows and Cols properties.

The examples below illustrate how the FormatString property works.

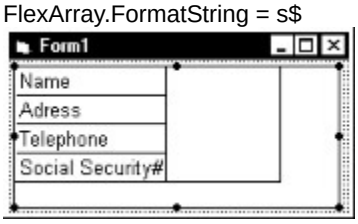
**** set column headers**

```
s$ = <Region      |<Product      |<Employee     |>Sales
FlexArray.FormatString = s$
```



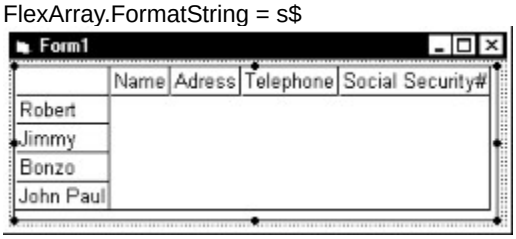
**** set row headers (note semicolon at start)**

```
s$ = ;Name|Adress|Telephone|Social Security#
FlexArray.FormatString = s$
```



**** set column and row headers**

```
s$ = |Name|Adress|Telephone|Social Security#
s$ = s$ + ;|Robert|Jimmy|Bonzo|John Paul
FlexArray.FormatString = s$
```



Data Type String

GridColor, GridColorFixed Properties

Description Determine the color used to draw the lines between FlexArray cells.

Usage [form!]FlexArray.**GridColor**[= *colorexpression*]
[form!]FlexArray.**GridColorFixed**[= *colorexpression*]

Remarks The GridColor property is used only when the GridLines property is set to *1 Lines*, and GridColorFixed is used only when GridLinesFixed is set to *1 Lines*. Raised and inset grids lines are always drawn in black and white.

Data Type Long (Color)

GridLines, GridLinesFixed Properties

Description Determines what type of lines should be drawn between cells.

Usage [form!]FlexArray.**GridLines**[= *setting*]

Setting The GridLines property settings are:

0 - None
1 - Lines (default for GridLines)
2 - Inset (default for GridLinesFixed)
3 - Raised

When the GridLines property is set to *1 - Lines,* the color of the lines is determined by the GridColor property.

Data Type Integer (Enumerated)

HighLight Property

Description Determines whether selected cells appear highlighted.

Usage [form!]FlexArray.**HighLight**[= *setting*]

Setting The HighLight property settings are:

0 - Never
1 - Always (default)
2 - With Focus

Remarks When this property is set to zero and the user selects a range of cells, there is no visual cue that shows which cells are currently selected.

Data Type Integer (Boolean)

LeaveCell Event

Description Occurs immediately before the currently active cell changes to a different cell.

Syntax Sub FlexArray_**LeaveCell** ( )

Remarks This event is useful if you want to implement cell-editing capabilities. In this case, you can trap this event to validate and apply changes to a cell before the user activates another cell .

The code below shows how this can be done. It assumes that there is an edit box called TextEdit that is used to edit the contents of the current cell.

```
Sub FlexArray_LeaveCell ()
   ' ** if the edit box is up, copy its contents
   If TextEdit.Visible Then
      FlexArray = TextEdit
      TextEdit.Visible = False
   End If
End Sub
```

See also the EnterCell event.

LeftCol Property

Description Determines the leftmost visible column (other than a fixed column) in the FlexArray. Not available at design time.

Usage [form!]FlexArray.**LeftCol**[= *numericexpression*]

Remarks You can use this property in code to scroll a FlexArray programmatically. Use the TopRow property to determine the topmost visible row in the FlexArray.

Data Type Integer

MergeCells Property

Description Determines whether cells with the same contents should be grouped in a single cell spanning multiple rows or columns.

Usage [form!]FlexArray.**MergeCells**[= *setting*]

Setting The MergeCells property settings are:

0 - Never (default)
1 - Free
2 - Restrict Rows
3 - Restrict Columns
4 - Restrict Both

Remarks The FlexArray cell merging technology allows you present data in a clear, appealing way. To see how cell merging is can be used in concert with FlexArrays sorting and column ordering capabilities, see the MERGE.MAK example on the distribution diskette.

To use FlexArrays cell merging capabilities, you must do two things:

1: Set MergeCells to a value other than zero. (The difference between the settings is explained below.)

2: Set the MergeRowcs_MergeColMergeRowProperties() and MergeCol() array properties to True for the rows and columns you wish to merge.

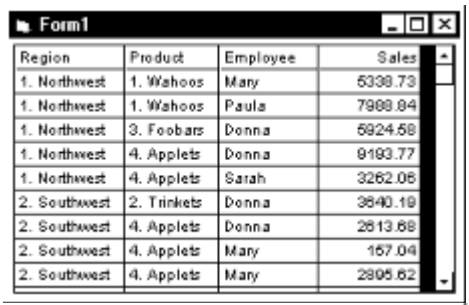
Thats all there is to it. FlexArray will merge cells with the same contents, and will update the merging automatically whenever you change the contents of any cells.

The difference between *Free* and *Restricted* merging is whether cells with the same contents should always be merged or only when adjacent cells to the left or to the top are also merged. This is hard to explain, but easy to show:

No Merging

MergeCells = 0
MergeRow(0) = True
MergeRow(1) = True
MergeRow(2) = True
MergeRow(3) = False

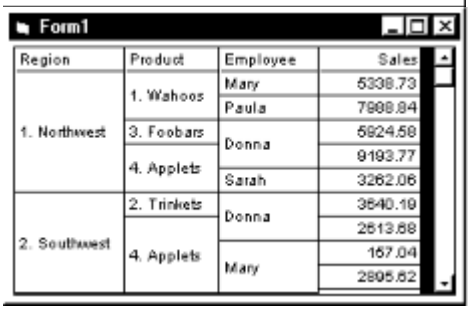
This is the regular spreadsheet view.



Free Merging

MergeCells = 1
MergeRow(0) = True
MergeRow(1) = True
MergeRow(2) = True
MergeRow(3) = False

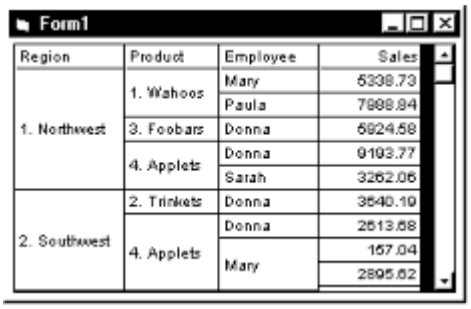
Notice how the third employee cell (Donna) merges across products to its left and across sales to its right.



Restricted Merging

MergeCells = 2
MergeRow(0) = True
MergeRow(1) = True
MergeRow(2) = True
MergeRow(3) = False

Notice how the third employee cell (Donna) no longer merges across sales.



Note When MergeCells is set to a value other than *0 - Never*, selection highlighting is automatically turned off. This is done mainly to speed up repainting, but also because selection of ranges containing merged cells may lead to counterintuitive results.

Data Type Integer (Enumerated)

MergeCol(), MergeRow() Properties

Description Determine which rows and columns should have their contents merged when the MergeCells property is set to a value other than *0 - Never*.

Usage [form!]FlexArray.**MergeRow**(*rownum*)[= { **True|False** }]
[form!]FlexArray.**MergeCol**(*colnum*)[= { **True|False** }]

Remarks If the MergeCells property is set to a non-zero value, adjacent cells with identical values are merged if they are in a row with the MergeRow property set to True or in a column with the MergeCol property set to True.

For details on how FlexArrays merging technology works, see the MergeCells property.

Data Type Integer (Boolean)

MouseCol, MouseRow Properties

Description Determine over which row and column the mouse pointer is. Not available at design time, read-only at run time.

Usage *rowvar* = [form!]FlexArray.**MouseRow**
colvar = [form!]FlexArray.**MouseCol**

Remarks You can use this property in code to determine where the mouse is and act accordingly. These properties are especially useful to display context-sensitive help on the contents of individual cells or to test whether the user has clicked on a fixed row or column.

Data Type Integer

Redraw Property

Description Enables or disables redrawing of the FlexArray control.

Usage [form!]FlexArray.**Redraw**[= { **True|False** }]

Remarks You can use this property in code to reduce flicker while making extensive updates to the contents of the FlexArray.

For example, the code below turns repainting off, makes several changes to the contents of the FlexArray, and then turns repainting back on to show the results:

```
 ** freeze FlexArray to avoid flicker
FlexArray.Redraw = False
 ** update FlexArray contents
For I% = FlexArray.FixedRows To FlexArray.Rows - 1
   FlexArray.TextArray(faIndex(i%, 1)) = GetName(i%, 1)
   FlexArray.TextArray(faIndex(i%, 2)) = GetName(i%, 2)
Next
 ** show results
FlexArray.Redraw = True
```

Data Type Integer

RemoveItem Method

Description Removes a row from a FlexArray control at run time

Syntax [form!]FlexArray.**RemoveItem** *index*

Remarks The RemoveItem method has these parts:

index
Integer representing the row to remove. To remove the first row, use index = 0.

Row, Col Properties

Description Determine the active cell in a FlexArray. Not available at design time.

Usage [form!]FlexArray.**Col**[= *colnum*]
[form!]FlexArray.**Row**[= *rownum*]

Remarks Use these properties to specify a cell in a FlexArray or to find out which row or column contains the current cell. Columns and rows are numbered from zero, beginning at the top for rows and at the left for columns.

Setting these properties automatically resets RowSel and ColSel, so the selection becomes the current cell. Therefore, to specify a block selection, you must set Row and Col first, then set RowSel and ColSel.

Note The Col, Row properties are not the same as the Cols, Rows properties.

Data Type Integer

RowColChange Event

Description Occurs when the currently active cell changes to a different cell.

Syntax Sub FlexArray_**RowColChange** ( )

Remarks This event occurs whenever the user clicks a cell other than the selected cell or when you programmatically change the active cell within a selection.

You can trigger this event in code by changing the current cell using the Col and Row properties.

The RowColChange event also occurs when a user clicks a new cell, but does not occur when you programmatically change the selected range without changing the active cell.

RowHeight() Property

Description Determines the height of the specified row in Twips. Not available at design time.

Usage [form!]FlexArray.**RowHeight**(*rownum*)[= *numericexpression*]

Remarks You can set RowHeight to zero to create invisible rows, or to -1 to reset the row height to its default value, which depends on the size of the current font.

If *rownum* is -1, then the FlexArray assumes you want to set the height of all rows at once.

Data Type Long

RowHeightMin Property

Description Allows you to specify a minimum row height for the entire control, in Twips.

Usage [form!]FlexArray.**RowHeightMin**[= *numericexpression*]

Remarks Use this property if you wish to use small fonts but want the rows to be tall. Setting this property is sometimes easier than setting individual row heights with the RowHeight() property.

Data Type Long

RowPosition(), ColPosition() Properties

Description Allow you to move rows and columns to specific positions on the FlexArray.

Usage [form!]FlexArray.**ColPosition(***colnum***)**[= *newcolnum*]
[form!]FlexArray.**RowPosition(***rownum***)**[= *newrownum*]

Remarks The index and setting must correspond to valid row or column numbers (in the range 0 to Rows - 1 or Cols - 1) or an error will be generated.

For example, the following code moves a column to first position when the user clicks on it:

```
Sub FlexArray_Click ()
   FlexArray.ColPosition(FlexArray.MouseCol) = 0
End Sub
```

When a row or column is moved with these properties, all formatting information moves with it, including width, height, alignment, colors, fonts, etc. To move text only, use the Clip property instead.

Data Type Integer

Rows, Cols Properties

Description Determine the total number of columns or rows in a FlexArray. The minimum is 0. The maximum number of rows and columns is limited by the memory available on your computer.

Usage [form!]FlexArray.**Cols**[= *numericexpression*]
[form!]FlexArray.**Rows**[= *numericexpression*]

Remarks You can use these properties to expand and shrink a FlexArray dynamically at run time.

Note The Cols, Rows properties are not the same as the Col, Row properties.

Data Type Integer

RowSel, ColSel Properties

Description Determine the starting or ending row or column for a range of cells. Not available at design time.

Usage [form!]FlexArray.**RowSel**[= *numericexpression*]
[form!]FlexArray.**ColSel**[= *numericexpression*]

Remarks You can use these properties to select a specific region of the FlexArray from code, or to read into code the dimensions of an area that the user selects.

The FlexArray cursor it the cell at **Row**, **Col**. The FlexArray selection is the region between rows **Row** and **RowSel** and columns **Col** and **ColSel**. Note that **RowSel** may be above or below **Row**, and **ColSel** may be to the left or to the right of **Col**.

Whenever you set the **Row** and **Col** properties, **RowSel** and **ColSel** are automatically reset so the cursor becomes the current selection. If you want to select a block of cells from code, you must set the **Row** and **Col** properties first, then set **RowSel** and **ColSel**.

Data Type Integer

Scroll Event

Description Occurs when the FlexArray scrolls its contents, either through the scroll bars, keyboard, or code changing the TopRow or LeftCol properties.

Syntax Sub FlexArray_**Scroll** ( )

Remarks You can use this event to perform calculations or to manipulate controls that must be coordinated with ongoing changes in scroll bars.

SelChange Event

Description Occurs when the selected range changes to a different cell or range of cells.

Syntax Sub FlexArray_**SelChange** ( )

Remarks The SelChange event occurs whenever the user clicks a cell other than the selected cell and as a user drags to select a new range of cells. A user can also select a range of cells by pressing the Shift key and using the arrow keys.

You can trigger this event in code by changing the selected region using the Row, Col, RowSel, or ColSel properties.

The RowColChange event also occurs when a user clicks a new cell but does not occur while a user drags the selection across the grid.

SelectionMode Property

Description Specifies whether a FlexArray should allow regular cell selection, selection by rows, or selection by columns.

Usage [form!]FlexArray.**SelectionMode**[= *setting*]

Setting The SelectionMode property settings are:

0 - Free (default)
1 - By Row
2 - By Column

Remarks Setting *0 - Free* allows selections to be made normally, spreadsheet-style. Setting *1 - By Row* forces selections to span entire rows, as in a multi-column list-box or record-based display. Setting *2 - By Column* forces selections to span entire columns, as if selecting ranges for a chart or fields for sorting.

Data Type Integer (Enumerated)

ScrollBars Property

Description Specifies whether a FlexArray has horizontal or vertical scroll bars.

Usage [form!]FlexArray.**ScrollBars**[= *setting*]

Setting The ScrollBars property settings are:

0 - None
1 - Horizontal
2 - Vertical
3 - Both (default)

Remarks Scroll bars appear on a FlexArray only if its contents extend beyond the FlexArrays borders. For example, a vertical scroll bar appears when the FlexArray can't display all of its rows. If ScrollBars is set to False, the FlexArray will not have scroll bars, regardless of its contents.

Note that if the FlexArray has no scroll bars in either direction, it will no allow *any* scrolling in that direction, even if the user uses the keyboard to select a cell that is off the visible area of the control.

Data Type Integer (Enumerated)

ScrollTrack Property

Description Specifies whether FlexArray should scroll its contents while the user moves the scroll box along the scroll bars.

Usage [form!]FlexArray.**ScrollTrack**[= { **True|False** }]

Remarks This property should normally be set to False to avoid excessive scrolling and flickering. Set it to True only if you want to emulate other controls that have this behavior.

Data Type Integer (Boolean)

Sort Property

Description Action-type property that sorts selected rows according to selected criteria. Not available at design-time, write-only at run time.

Usage [form!]FlexArray.**Sort**[= *setting*]

Setting The Sort property settings are:

0 - None	
1 - Generic Ascending	guesses whether text is string or number
2 - Generic Descending	
3 - Numeric Ascending	converts strings to numbers
4 - Numeric Descending	
5 - String Ascending	case-insensitive string comparison
6 - String Descending	
7 - String Ascending	case-sensitive string comparison
8 - String Descending	
9 - Custom	uses Compare event to compare rows

Remarks The Sort property moves always sorts entire rows. The range to be sorted is specified by setting the Row and RowSel properties. If Row and RowSel are the same, FlexArray assumes that you want to sort all non-fixed rows.

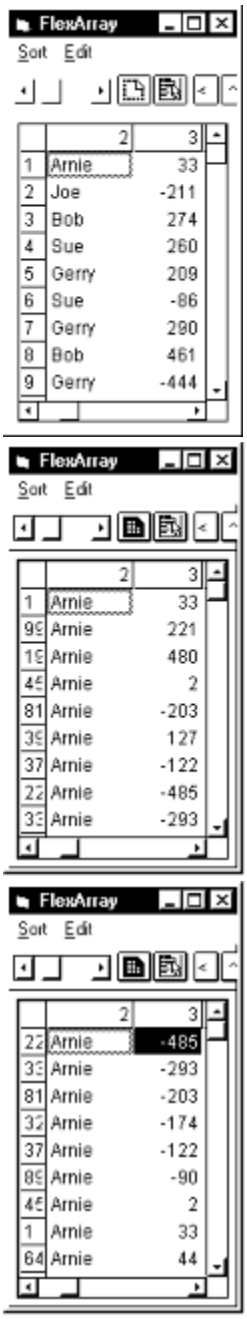
They keys used for sorting are determined by the Col and ColSel properties, always from the left to the right. For example, if Col = 3 and ColSel = 1, the sort would be done according to the contents of columns 1, then 2, then 3.

The method used to compare the rows is determined by the setting, as explained above. The *9 - Custom* setting is the most flexible, since it fires the Compare event that allows you to compare rows in any way you want, using any columns in any order (see the Compare event for details). However, this method is also much slower that the others, typically by a factor of ten, so it should be used only when really necessary.

An alternative to using the *9 - Custom* setting is to create an invisible column, fill it with the keys, then sort based on it with one of the other settings. This is a very good approach for sorting based on dates, for example.

The code and pictures below shows how the sort property works:

```
 ** fill FlexArray with random data (left picture)
For i% = FlexArray.FixedRows to FlexArray.Rows - 1
   FlexArray.TextArray(faIndex(i%, 1)) = RandomName()
   FlexArray.TextArray(faIndex(i%, 2)) = RandomNumber()
Next
 ** sort by name (center picture)
FlexArray.Row = 1
FlexArray.Col = 1
FlexArray.Sort = 1   Generic Ascending
 ** sort by name and number (right picture)
FlexArray.Row = 1
FlexArray.Col = 1
FlexArray.ColSel = 2
FlexArray.Sort = 1   Generic Ascending
```



Data Type Integer (Enumerated)

Text Property

Description Sets or retrieves the text contents of a cell or range of cells.

Usage [form!]FlexArray.**Text** [= *string*]

Remarks When retrieving, the Text property always retrieves the contents of the current cell, defined by the Row and Col properties.

When setting, the Text property sets the contents of the current cell or of the current selection, depending on the setting of the FillStyle property.

See also the TextArray property.

Data Type String

TextArray() Property

Description Sets or retrieves the text contents of an arbitrary cell.

Usage [form!]FlexArray.**TextArray(***cellindex***)** [= *string*]

Remarks This property allows you to set or retrieve the contents of a cell without changing the Row and Col properties.

The *cellindex* parameter determines which cell to use. It is calculated by multiplying the desired row by the Cols property and adding the desired column. The clearest and most convenient way to calculate *cellindex* is to define a function to do it, as show below:

```
 ** calculate index for use with TextArray property
Function faIndex(row%, col%) As Long
     faIndex = row * FlexArray.Cols + col
End Function

Sub Form_Load()
Dim i%
 ** fill FlexArray with data using TextArray property
For i% = FlexArray.FixedRows to FlexArray.Rows - 1
    ** column 1
  FlexArray.TextArray(faIndex(i%, 1)) = RandomName()
    ** column 2
  FlexArray.TextArray(faIndex(i%, 2)) = RandomNumber()
Next
```

See also the Text property.

Data Type String

TextStyle, TextStyleFixed Properties

Description Allows you to specify 3D effects for displaying text. TextStyle determines the style of regular FlexArray cells, and TextStyleFixed determines the style of fixed rows and columns.

Usage [form!]FlexArray.**TextStyle** [= *setting*]
[form!]FlexArray.**TextStyleFixed** [= *setting*]

Setting The TextStyle and TextStyleFixed property settings are:

0 - Flat
1 - Raised
2 - Inset
3 - Raised Light
4 - Inset Light

Remarks Settings 1 and 2 work best for large and bold fonts. Settings 3 and 4 work best for small regular fonts.

See also the CellTextStyle property.

Data Type Integer

TopRow Property

Description Determines the uppermost row displayed in the FlexArray. Not available at design time.

Usage [form!]FlexArray.**TopRow**[= *numericexpression*]

Remarks You can use this property in code to programmatically read or set the visible top row of the FlexArray. Use the **LeftCol** property to determine the leftmost visible column in the FlexArray.

When setting this property, the largest possible row number is the total number of rows minus the number of rows that can be visible in the FlexArray. Attempting to set TopRow to a greater row number will cause the FlexArray to set it to the largest possible value.

Data Type Integer

Version Property

Description This property returns the version of the VSFlex controls currently loaded in memory.

Usage *checkversion* = [form!]FlexArray.**Version**

Remarks You may want to check this value at the Form_Load event, to make sure the version that is executing is at least as current as the version used to develop your application.

The version number is a three digit integer where the first digit represents the major version number and the last two represent the minor version number. For example, version 1.23 would return 123.

This property is read-only.

Data Type Integer

WordWrap Property

Description This property determines whether text within a cell should be broken between words if a word would extend past the edge of the cell. Return characters Chr(13) also force line breaks.

Usage [form!]FlexArray.**WordWrap** [= {**True|False**}]

Remarks FlexArray can display text slightly faster if you set WordWrap to False.

Data Type Integer (Boolean)

Using the FlexString Control

FlexString is a control that allows you to incorporate regular-expression text matching into your VB programs. This allows you to easily parse complex text input or to offer regular expression search and replace features such as those found in professional packages such as Microsoft Word, Visual C++, and Visual Basic.

FlexString looks for text patterns on its Text property, and lets you inspect and change the matches it found. The text patterns are specified through the Pattern property, using regular expressions. The syntax for regular expressions is described below.

Matching

As soon as you assign a string to the Text or Pattern properties, FlexString tries to find as many matches as it can, and returns the number of matches found in the MatchCount property. You can then scan through the matches by changing the MatchIndex property and reading the MatchString property.

For example, the following code would scan a string and print all phone numbers in the San Francisco area (the phone pattern used in the example is not very flexible, but its good enough to show how the control works):

```
Dim i%
Dim PhonePat$

FlexString.Text = ClientList$
PhonePat = (415)[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
FlexString.Pattern = PhonePat
Debug.Print FlexString.MatchCount   match(es) found.
For i% = 0 to FlexString.MatchCount - 1
   FlexString.MatchIndex = I
   Debug.Print FlexString.MatchString
Next
```

Replacing

You can also replace matches automatically, using the Replace property. For example, say you wanted to change all instances of the (415) area code with (510):

```
Dim PhonePat$

FlexString.Text = ClientList$
PhonePat = (415)
FlexString.Pattern = PhonePat
Debug.Print FlexString.MatchCount   match(es) found.
FlexString.Replace = (510)
```

When a string is assigned to the **Replace** property, FlexString immediately replaces all matches with the new string.

Tag Matches

The FlexString control also allows you to do tagged matches. By tagging matches, you can easily determine what parts of the string matched what parts of the pattern.

For example, say you wanted to make your letters more informal by replacing all occurrences of Mr. John Doe, Ms Jane Doe, and Mrs. Penny Doe, with John, Jane, and Penny. Heres the code to do it:

```
FlexString.Text = ClientList$
FlexString.Pattern = Mr\.? {[A-Za-z]+} {[A-Za-z]+}
FlexString.Replace = {0}
FlexString.Pattern = Ms\.? {[A-Za-z]+} {[A-Za-z]+}
FlexString.Replace = {0}
FlexString.Pattern = Mrs\.? {[A-Za-z]+} {[A-Za-z]+}
FlexString.Replace = {0}
```

The curly brackets mark the tagged parts of the pattern. In this example, there are two tags, {0} and {1}, that match the persons first and last names. The first tag is used in the replace string to retrieve the persons first name.

Regular expressions

A regular expression is a notation for specifying and matching strings. Like an arithmetic expression, a regular expression is a basic expression or one created by applying operators to component expressions.

FlexString has a string property called Pattern that holds a regular expression. FlexString recognizes the following special characters in Pattern:

Char	Meaning
^	Matches the beginning of a string.
$	Matches the end of a string.
.	Matches any character.
[ ]	Character class, or complemented character class if the first character inside the brackets is a caret (^).
*	Repeat previous zero or more times.
+	Repeat previous one or more times.
?	Repeat previous zero or one time.
\	Escape next character.
{ }	Tagged match.

The following examples illustrate these:

Pattern	Matches
^stuff	strings that start with stuff.
stuff$	strings that end with stuff.
^...$	any 3-character string.
[AEIOU]	any uppercase vowel.
[0-9]	any digit.
[AZaz][0-9]	any letter followed by any digit.
[^0-9]	any character except a digit.
[A-Z][0-9]*	any upper-case letter optionally followed by any number of digits.
[A-Z][0-9]+	any upper-case letter followed by at least one digit.
[A-Z][0-9]?	any upper-case letter optionally followed by one digit.
[+-]?[0-9]+	any integer optionally preceded by a sign.
[+-]?[0-9]+\.?[0-9]*	any real number.

FlexString Control Reference

Description	The FlexString control is a powerful regular expression engine. With FlexString, you can define, find and replace patterns in strings.
Remarks	Use FlexString it to provide regular expression search-and-replace capabilities similar to those available in professional packages such as Word, Visual C++, or Visual Basic. Or use it to parse input strings in complex formats.
File Name	VSFLEX.VBX
Object Type	FlexString
Note	Before you can use a FlexArray control in your application, you must add VSFLEX.VBX to your project (see the Visual Basic manual for details). To automatically include VSFLEX.VBX in new projects, put it in an AUTOLOAD.MAK file. When distributing your application, you should install the VSFLEX.VBX file in the user's Microsoft Windows SYSTEM subdirectory.

FlexString Summary

Properties (default: Text)

(About)	* Error	Left
* MatchCount	* MatchIndex	* MatchLength
* MatchStart	* MatchString	Name
Parent	* Pattern	* Replace
Tag	* TagCount	* TagIndex
* TagStart	* TagLength	* TagString
* Text	Top	* Version

Error Property

Description Returns status information after you set the Pattern or Text properties. Read-only.

Usage *err* = [form!]FlexString.**Error**

Setting The settings returned by the Error property are:

0 - None
1 - Out of memory
2 - Unbalanced '[' in pattern
3 - Unbalanced '{' in pattern
4 - No valid pattern
5 - Bad TagIndex in replacement string
6 - No match
7 - MatchIndex too high

Remarks You should always check the Error property when a match fails.

Error 1 - Out of Memory will occur if you assign a string that is too long to the Text property or a pattern that is to complex to the Pattern property. This should rarely occur, since FlexString can handle strings with up to 32,000 characters.

Errors *2 - Unbalanced '[' in pattern* and *3 - Unbalanced '{' in pattern* occur when you assign invalid patterns to the Pattern property. If you really want to match brackets, remember to escape them with the backslash character (i.e. use \{ instead of {).

Error *4 - No valid pattern* occurs when you try to retrieve the results of a match when the Pattern or Text properties are empty.

Error *5 - Bad TagIndex in replacement string* occurs when you use a tag in a replacement string for which there is no match (i.e. Pattern = {[a-z]*} , Replace = {0} {1}).

Error *6 - No match* occurs when you try to retrieve the results of a match and the match failed.

Error *7 - MatchIndex too high* occurs when you try to select a match greater then or equal to MatchCount.

Data Type Integer (Enumerated)

MatchCount Property

Description Returns the number of matches found after you set the Pattern or Text properties. Read-only.

Usage *count* = [form!]FlexString.**MatchCount**

Remarks You can retrieve information about each match by setting the MatchIndex property to a value between 0 and MatchCount - 1 and reading the MatchLength, MatchStart, and MatchString properties.

Data Type Integer

MatchIndex Property

Description Determines the current match.

Usage [form!]FlexString.**MatchIndex**[= *numericexpression*]

Remarks You can retrieve information about the current match by reading the MatchLength, MatchStart, and MatchString properties.

The MatchIndex property can range from 0 to MatchCount - 1.

Data Type Integer

MatchLength Property

Description Returns the length of the current match, in characters. Read-only.

Usage *varlen* = [form!]FlexString.**MatchLength**

Remarks You can retrieve information about the current match by reading the MatchLength, MatchStart, and MatchString properties.

Data Type Integer

MatchStart Property

Description Returns the position of the current match within the Text property string, starting from zero. Read-only.

Usage *varstart* = [form!]FlexString.**MatchStart**

Remarks You can retrieve information about the current match by reading the MatchLength, MatchStart, and MatchString properties.

Data Type Integer

MatchString Property

Description Sets or returns the string corresponding to the current match.

Usage [form!]FlexString.**MatchString**[= *string*]

Remarks If you assign a new string to the MatchString property, FlexString will modify the string in the Text property and will attempt to do a new match.

Data Type String

Pattern Property

Description Sets or returns the regular expression being used for matching.

Usage [form!]FlexString.**Pattern**[= *string*]

Remarks The syntax for the Pattern property is described in detail in the FlexString QuickStart section of this document.

Data Type String

Replace Property

Description Replaces all matches with a specified string.

Usage [form!]FlexString.**Replace**[= *string*]

Remarks The replacement occurs as soon as you assign the new text to the Replace property. If you wish to perform the replacement on several strings, you need to assign the replacement string once for each string, as shown below:

```
FlexString.Pattern = hte
While Not EOF(1)
   FlexString.Text = GetLine (1)
   FlexString.Replace = the
Wend
```

The Replacement string may contain *tag replacements*, specified using curly brackets. The tag replacements expand into the matched text. For example:

```
FlexString.Pattern = "{[A-Za-z]+}\.{...}"
FlexString.Text = AUTOEXEC.BAT
FlexString.Replace = File {0}.{1}, Name: {0}, Ext: {1}
Debug.Print FlexString.Text
   File AUTOEXEC.BAT, Name: AUTOEXEC, Ext: BAT
```

Note how the first dot in the Pattern is escaped with the backslash character to really match a dot. The remaining dots match any character.

Data Type String

TagCount Property

Description Returns the number of tags found after you set the Pattern, Text, or MatchIndex properties. Read-only.

Usage *count* = [form!]FlexString.**TagCount**

Remarks You can retrieve information about each tag by setting the TagIndex property to a value between 0 and TagCount - 1 and reading the TagLength, TagStart, and TagString properties.

Tags are defined by enclosing parts of the regular expression string in the Pattern property between curly brackets. For example:

```
FlexString.Text = Mary had a little lamb
FlexString.Pattern = Mary had {.*}
Debug.Print FlexString.TagCount; FlexString.TagIndex;
Debug.Print [ + FlexString.TagString + ]
   1   0 [a little lamb]
```

Data Type Integer

TagIndex Property

Description Determines the current tag.

Usage [form!]FlexString.**TagIndex**[= *numericexpression*]

Remarks You can retrieve information about the current tag by reading the TagLength, TagStart, and TagString properties.

The TagIndex property can range from 0 to TagCount - 1.

For example:

```
FlexString.Text = Mary had a little lamb
FlexString.Pattern = {[^ ]*} had {.*}
FlexString.TagIndex = 0
Debug.Print [ + FlexString.TagString + ]
  [Mary]
FlexString.TagIndex = 1
Debug.Print [ + FlexString.TagString + ]
  [a little lamb]
```

Data Type Integer

TagLength Property

Description Returns the length of the current tag, in characters. Read-only.

Usage *varlen* = [form!]FlexString.**TagLength**

Remarks You can retrieve information about the current tag by reading the TagLength, TagStart, and TagString properties.

Data Type Integer

TagStart Property

Description Returns the position of the current tag within the Text property string, starting from zero. Read-only.

Usage *varstart* = [form!]FlexString.**TagStart**

Remarks You can retrieve information about the current tag by reading the TagLength, TagStart, and TagString properties.

Data Type Integer

TagString Property

Description Sets or returns the string corresponding to the current tag.

Usage [form!]FlexString.**TagString**[= *string*]

Remarks If you assign a new string to the TagString property, FlexString will modify the string in the Text property and will attempt to do a new match.

Data Type String

Text Property

Description Sets or returns the text to be scanned searching for the Pattern string.

Usage [form!]FlexString.**Text**[= *string*]

Remarks FlexString will try to perform a match as soon as you assign a string to the Text or to the Pattern properties.

To find out how many matches were found, read the MatchCount property.

To retrieve information about each match, set the MatchIndex property to a value between 0 and MatchCount - 1 and read the MatchLength, MatchStart, and MatchString properties.

Data Type String

Version Property

Description This property returns the version of the VSFlex controls currently loaded in memory.

Usage *checkversion* = [form!]FlexString.**Version**

Remarks You may want to check this value at the Form_Load event, to make sure the version that is executing is at least as current as the version used to develop your application.

The version number is a three digit integer where the first digit represents the major version number and the last two represent the minor version number. For example, version 1.23 would return 123.

This property is read-only.

Data Type Integer

VideoSoft Products

To get an order form, click **HERE**.

VSVBX

A set of three custom controls for interface design and text parsing.

Icon	Name	Object	Description
	Elastic	vsElastic	Smart containers that resize themselves and their child controls, automatically create labels and 3-D frames for its child controls, and can also be used as progress indicators and labels.
	IndexTab	vsIndexTab	Allows you to group controls by subject, using the familiar notebook metaphor that has become a Windows standard.
	Awk	vsAwk	Parsing engine named and patterned after the popular Unix utility, plus a powerful expression evaluator.

VSVIEW

A set of four custom controls for creating, viewing, and printing text and graphics.

Icon	Name	Object	Description
	InForm	vsInForm	A control that you can drop into any container to customize its title bar, frame, resizing behavior, and frame buttons. InForm also allows you to monitor the clipboard, drag and drop files from File manager, and more.
	Printer	vsPrinter	A much improved printer object with word wrap, headers and footers, multi-column printing, graphics, and multi-page Print Preview capability.
	ViewPort	vsViewPort	A control that gives you a scrollable virtual area so you can fit more controls in your windows. Great for implementing Print Preview and programs that look like the Program Manager.
	Draw	vsDraw	A versatile drawing control that lets you create complex images, view them on the screen, copy them to the clipboard, or print them. Great for technical drawings, maps, and diagrams.

VSFLEX

A set of two custom controls for analyzing, formatting, and displaying information.

Icon	Name	Object	Description
	FlexArray	vsFlexArray	A new way to display and operate on tabular data. FlexArray gives you total flexibility to display, sort, merge, and format tables containing strings and pictures.
	FlexString	vsFlexString	A powerful regular expression engine. With FlexString, you can find and replace patterns in strings. Use it to provide regular expression search-and-replace capabilities or to parse input strings.

CODEBOOK

A handy, integrated suite of utilities to help you develop Visual Basic applications.

Order Form

(You may print this form by selecting the File|Print command).

TO: VideoSoft
2625 Alcatraz Avenue, Suite 271
Berkeley, California 94705

To order by phone, call
(800) 547-7295 (from within the US)
(510) 704-8200 (from anywhere)
(510) 843-0174 (fax)

Please register my copy of the following VideoSoft products. I am enclosing a check or money order for the amount of:

VSVBX Single developer	US$ 45.00	
Additional developers	__ x 45.00	
VSVIEW Single developer	99.00	
Additional developers	__ x 99.00	
VSFLEX Single developer	99.00	
Additional developers	__ x 99.00	
CODEBOOK Single developer	45.00	
Additional developers	__ x 45.00	
Shipping and Handling Domestic	6.00	
Shipping and Handling International	10.00	
CA Sales Tax (CA residents only)	8.5%	
TOTAL		US$

Note: Call us for details on site licenses and volume discounts.

Name:

Company:

Street:

City, State, ZIP:

Country:

Phone:

Where did you hear about the VideoSoft products?